UNIVERSIDAD POLITÉCNICA
DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INFORMÁTICOS

MÁSTER UNIVERSITARIO EN INGENIERÍA DEL SOFTWARE –
EUROPEAN MASTER IN SOFTWARE ENGINEERING

Mining Mobile Apps Reviews to Support Release Planning

Master Thesis

Lorenzo Villarroel Pérez

Madrid, July 2015

This thesis is submitted to the ETSI Informáticos at Universidad Politécnica de Madrid in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering.

Master Thesis
Master Universitario en Ingeniería del Software – European Master in Software Engineering
Thesis Title: Mining Mobile Apps to Support Release Planning
Thesis no: EMSE-2015-09
July 2015

Author: Lorenzo Villarroel Pérez
European Master in Software Engineering
Universidad Politécnica de Madrid

Supervisor:

Gabriele Bavota
Doctor
University of Salerno (Italy)

Faculty of Computer Science
Free University of Bozen-Bolzano

Co-supervisor:

Ernestina Menasalvas
Doctor
Universidad Politécnica de Madrid

DLSIIS
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

ETSI Informáticos
Universidad Politécnica de Madrid
Campus de Montegancedo, s/n
28660 Boadilla del Monte (Madrid)
Spain

Acknowledgements

I want to personally thank my supervisor from Free University of Bozen-Bolzano, Gabriele Bavota for all his support during this thesis. His advises about how to work with everything related to a machine-learner made this work possible, and thanks to my co-supervisor from Universidad Politécnica de Madrid (UPM), Ernestina Menasalvas for her support on the thesis. Also, thanks to Abimael Barea, student from UPM to provide us with an example of file with reviews that you can download from Google Play Developer Console and his reviews about the graphical interface of the tool created.

CONTENT

1 ABSTRACT

The mobile apps market is a tremendous success, with millions of apps downloaded and used every day by users spread all around the world. For apps' developers, having their apps published on one of the major app stores (e.g. Google Play market) is just the beginning of the apps lifecycle. Indeed, in order to successfully compete with the other apps in the market, an app has to be updated frequently by adding new attractive features and by fixing existing bugs. Clearly, any developer interested in increasing the success of her app should try to implement features desired by the app's users and to fix bugs affecting the user experience of many of them. A precious source of information to decide how to collect users' opinions and wishes is represented by the reviews left by users on the store from which they downloaded the app. However, to exploit such information the app's developer should manually read each user review and verify if it contains useful information (e.g. suggestions for new features). This is something not doable if the app receives hundreds of reviews per day, as happens for the very popular apps on the market.

In this work, our aim is to provide support to mobile apps developers by proposing a novel approach exploiting data mining, natural language processing, machine learning, and clustering techniques in order to classify the user reviews on the basis of the information they contain (e.g. useless, suggestion for new features, bugs reporting). Such an approach has been empirically evaluated and made available in a web-based tool publicly available to all apps' developers. The achieved results showed that the developed tool: (i) is able to correctly categorise user reviews on the basis of their content (e.g. isolating those reporting bugs) with 78% of accuracy, (ii) produces clusters of reviews (e.g. groups together reviews indicating exactly the same bug to be fixed) that are meaningful from a developer's point-of-view, and (iii) is considered useful by a software company working in the mobile apps' development market.

2 INTRODUCTION

The mobile apps world is a very competitive environment, featuring millions of apps sold in popular online markets like Google Play or the Apple App store. Publishing an app in such markets literally means reaching millions of potential customers spread around the world. However, the life cycle of mobile apps does not end with their availability to customers. Indeed, developers continuously need to improve their apps in order to increase sales and, as a consequence, their revenues.

When a developer wants to improve his app, there are several approaches he can adopt. Firstly, he can rely on massive software testing to spot bugs and fix them. However, despite the effort invested, testing will never exercise an app as millions of users using it in different environments and on heterogeneous mobile devices. Second, a developer could get feedback from apps' users by performing surveys. This means designing a survey, make sure that a representative subset of the apps' users population answers it, and analyse the results to obtain precious insights on how to evolve the apps in order to maximise its success. The effort besides this process is not negligible. A third possibility is to exploit a precious source of information present in all apps' markets: *the users' reviews*. In particular, users downloading a specific app can leave a comment (e.g. feedback) to the app's developers by (i) assigning a rating to the app (generally expressed as a number of stars going from one to five), and (ii) write in a free-text field to report bugs, recommending new features, or simply describe their feelings while using the app.

Those reviews are written in a colloquial and natural language and represent a very precious source of information for apps' developers. In order to effectively get advantage of the information that users leave on the reviews, currently a developer has to navigate through the entire set of reviews left by users and read them manually. Some apps' market, like Google Play, provide some kind of support to the developers, like the possibility to rank the reviews by rating to easily spot

the more critical ones. However, as shown by Crider [Crider, 2014], some users just assign very low ratings (e.g. one star) just to increase the visibility of their review that, in many cases, is more positive than negative in its contents. This clearly reduces the visibility of other reviews that could potentially contain information about bugs or features, hiding them from the developers' eyes. Thus, this trivial support provided by the mobile apps' markets is not sufficient at all.

In this thesis, we will develop a novel approach, integrated in a publicly available tool, to automatically analyse the reviews that users post in the mobile apps' markets. While the developed approach is general, we instantiate our problem on Google Play market (e.g. we provide support to Android developers). Basically, we want to exploit users' reviews to recommend the developers on how to improve their app and understand what the users need [Mickel, 2010], how users are using the application and what problems they are experiencing, and their satisfaction using the app [UserZoom, 2014]. This information can be very useful for developers when releasing a new version of their app. On the other hand, around 70% of the user feedback is not useful for the developers [JoJo, 2011], so creating a way that helps developers to automatically distinguish between useful and not useful information in the user feedback, will decrease the time wasted in analysing this feedback and will short the time between version release.

2.1 Objectives and Result

The objective of this thesis is to develop a tool supporting the developers in gathering useful information from the reviews left by users of their apps. In particular, we aim at automatically categorising user reviews in three categories: suggestions for new features, bugs reporting, and other. Furthermore, inside those three categories, the reviews will be clustered together based on their specific topic (e.g. all reviews reporting the same bug will be clustered together).

In a nutshell, the three main steps we followed to reach our objective are the following:

- Develop an approach to automatically categorise user reviews. This approach is composed by the following sub-steps:
    - Pre-processing of reviews, where punctuation signs are removed and text is converted into lower-case.
    - N-grams extractions, where we extract the different combinations of 1,2,3,4 words contained in the review text.
    - Stop-words and stemming, where we remove the stop-words contained in the 1-grams and we transform those to their root words (stemming)
    - Managing negations, where we eliminate possible negations of key words that determine one category in order to avoid misclassifications
    - Merging synonyms, where we merge all the words that we considered synonyms into a common word in order to increase the efficiency of the classification
    - Creating training set for machine learner, where we construct the set of reviews necessary to get the machine-learner working
- Develop an approach to cluster related user reviews: we exploit Information Retrieval techniques and clustering algorithms to automatically cluster together related reviews (e.g. those indicating the same bug).
- Implementing the web-based tool, allowing mobile apps' developers to take advantage of the approaches we developed.

Results of the evaluation described in section 5 show that in 78% of the cases, the tool is able to correctly classify user reviews, which means that it has a reasonable accuracy. Furthermore, the cluster of reviews automatically generated by the tool have, in all cases, a similarity over 70% respect the clusters manually generated by industrial developers, which shows the meaningfulness of the clusters of reviews that the tool is able to generate.

Finally, feedback received by a project manager of a mobile apps' company confirms the usefulness of our tool in an industrial scenario.

2.2 Structure of the Document

The document is structured in the following way:

- The current chapter is introductory and it provides a general vision about what is the work related to and its objectives
- The third chapter will provide an overview of the context of the Project and it will discuss about the already existing work on the field
- The fourth chapter is dedicated to the tool created to categorise and cluster the reviews
- The fifth chapter details the description and the results of the evaluations performed to measure the efficiency of the tool
- The sixth chapter contains the conclusions of all the development process and the future work that could be done

3 BACKGROUND

This chapter describes the more relevant work carried out by other researchers in the software engineering field.

3.1 Identification and Classification of Requirements from App User Reviews

Authors: Hui Yang, Peng Liang

The authors propose an approach that is able to identify and classify requirements from user reviews, into functional and non-functional requirements. In order to do that, they use a keyword extraction technique (TF-IDF) and NLP technique (regular expression). They carried out an experiment in order to evaluate the results of the classification, using 1000 user reviews from iBooks App in the English Store. They noticed that when provided an appropriate size of sample reviews, their approach achieves a stable precision of requirements classification, which is meaningful and practical for app developers that want to elicit requirements from users.

In contrast with the work presented in this thesis, this approach does not use any machine learning techniques and it relies on manually constructed regular expressions. This clearly limits its generalisability. Also, it only detects functional and non-functional requirements, it does distinguish bugs reporting from suggestions for new features, and does not group the reviews by topics.

They divide the classification of the reviews into two steps: User reviews extractor and requirements identifier and classifier

User Reviews Extractor

In order to extract reviews from the stores, they use the APIs provided by an open source package AppReviews. In order to use those APIs, they need the user to

input the user to input the URL ID of the app and the APP COUNTRY ID of the language that they want to retrieve the reviews.

Requirements Identifier and Classifier

They propose a process divided in 5 phases that takes as an input the reviews previously extracted and produce as output the requirements extracted from the reviews.

Phase 1: Input Reviews to be Processed.

In this phase they collect and write in specific format the user reviews extracted from the user reviews extractor.

Phase 2: Pre-process User Reviews

In this phase they pre-process the extracted user reviews by combining the title and the text of the reviews, eliminating punctuation marks and stop words, filtering spam reviews and word stemming.

Phase 3: Extract Keywords

In this phase they extract the keywords required to identify requirements. To do so, they manually classified a certain amount of reviews as functional and non-functional requirements, which they considered as correct classification, and used them to extract automatically the keywords from them using TF-IDF technique.

Phase 4: Combine Keywords: Requirements Identifier and Classifier

In this phase they combine the extracted keywords into in various logical relationships of regular expressions.

Phase 5: Identify and Classify User Reviews

In this phase they identify user requirements from the pre-processed content of reviews (phase 2) using the regular expressions (phase 4).

In conclusion, the authors present an approach that automatically classifies and identifies requirements from user reviews. They validated the proposed approach with user reviews collected from a popular app in Apple's English App Store called iBooks. The results of the validation show that when their approach receives an appropriate size of sample reviews, they can achieve a stable precision of reviews classification, in particular for non-functional requirements.

3.2 AR-Miner: Mining Informative Reviews for Developers from Mobile App Marketplace

Authors: Ning Chen, Jialiu Lin, Steven C. H. Hoi, Xiaokui Xiao, Boshen Zhang

The authors present a framework for App Review Mining which given a set of user reviews, shows the user the most informative reviews grouped and prioritised by a ranking scheme. They carried out an evaluation with reviews of four popular Android apps to evaluate the approach and the approach shows the most informative reviews in an efficient and effective way.

In contrast with the work presented in this thesis, AR-Miner shows the reviews that may be informative for the developers, grouped and ordered by the importance of the information they contain, letting the developer to make an extra effort to get to know the category of this useful information (e.g. bug, feature, etc.).

They divide the filter of informative reviews, grouping and prioritisation of the reviews into five steps: pre-processing, filtering, grouping, ranking, and visualisation.

Pre-processing

In this step they split the text of the review into several sentences using a standard sentence splitter. They did it because within a review containing several sentences, not all of them have to be necessarily informative. Furthermore, they tokenise the

resulted sentences by removing all non-alpha-numeric characters, all punctuation signs, stop words and rare words, and then they stemmer to the resulting words.

Filtering

In this step, they filtered out the non-informative reviews. To do so, they introduce two class labels: {informative, non-informative}. Then they use a machine-learning classificator that works with an already created historical training data. The machine -earning algorithm they use is called Expectation Maximisation for Naive Bayes (EMNB). Then, with the classificator already built they can assign a label to any unlabelled review and then filter out the ones labelled as non-informative.

Grouping

In this step the authors partition the remaining review instances into several groups according to how semantically similar are the reviews from one group to another. They use a technique called "topic modelling" which assigns multiple topics to each review. They did that because a review containing multiple phrases might discuss different topics.

Ranking

In this step the authors order the created group of reviews, and each review inside every group, by their relative importance. To do so, they developed a flexible ranking algorithm that ranks the group of reviews and the reviews by their importance.

Visualization

In this step the authors provide a way to visualise the results of their ranking model. To achieve that they create a radar chart of the top n groups of categories labelled by the most representative group of words of the groups, and in order to see the complete set of reviews you can click on each label of the groups and it will show a list with all the reviews contained in this group.

3.3 Analysis of User Comments: An Approach for Software Requirements Evolution

Authors: Laura V. Galvis Carreño and Kristina Winbladh

The authors propose an approach that analyses a set of reviews and extract from them the main topics contained on the reviews along with some representative sentences of those topics. To do so, they used information retrieval techniques to extract common topics and present users' opinions about those topics. They carried out an experiment with 3 different apps and reviews, and they observed that the automatically retrieved topics matched the manually retrieved topics that they extracted previously.

In contrast with the work presented in this thesis, this approach just extracts the common topics of the reviews and groups them, which is similar to the clustering step that we perform, but the authors neither categorise reviews into features or bugs nor order them by their importance.

They divide the extraction of the topics into three steps: Input Data, Text Analysis and User Comment Report.

Input Data

In this step the authors perform the acquisition and preparation of data. They use reviews from several applications of the Android Marketplace, but it could come from anywhere else. The preparation of data consists of various steps: Tokenise, where they split the reviews into tokens that are the resulting words by splitting the review by the common sentence delimiters, Change to Lowercase, where they convert all the tokens into lowercase and Remove Noise, where they add a negative connotation to words that are placed near the word "not" and removing the stop words, and also removing empty reviews after the removing noise process.

Text Analysis

In this step the authors analyse the input data in order to find topics associated to each sentence. They adapt the Aspect and Sentiment Unification Model (ASUM), which incorporates both topic modelling and sentiment analysis in order to be able to use it with the reviews.

User Comment Report

In this step the authors provide a report with meaningful feedback to the developers. The report presents the classified information, organised by topics and sentiments, including two or three possible labels for each topic-sentiment. The labels are the words with the higher probability to belong to the topic-sentiment group.

3.4 How Can I Improve My App? Classifying User Reviews for Software Maintenance and Evolution

Authors: S. Panichella, A. Di Sorbo, E. Guzman, C. A.Visaggio, G. Canfora and H. C. Gall

The authors propose an approach that takes advantage of three techniques: Natural Language Processing, Text Analysis and Sentiment Analysis in order to automatically classify app reviews into categories that are relevant to software maintenance and evolution. They carried out an experiment with seven apps that were in the list of the most popular apps in the year 2013 and they observed that by combining the three techniques they were achieving a better precision when categorising reviews than using those three techniques individually.

In contrast with the work presented in this thesis, this approach by Panichella et al. categorises the reviews into various categories using machine learning techniques, which is similar to the step we perform to categorise reviews into features and

bugs. However, inside the categorisation, this approach does not organise the reviews into clusters with the same topic in common.

They divided the process of categorising reviews automatically into four steps: Taxonomy for Software Maintenance and Evolution, Feature Extraction, Learning Classifiers and Evaluation.

Taxonomy for Software Maintenance and Evolution

In this step the authors analysed users reviews of seven Apple Store and Google Play apps and rigorously deduced a taxonomy of the reviews containing useful content for software maintenance and evolution. They analysed the reviews with a sentence-level granularity, because in a review some sentences may be relevant for developers and some others not. The output of this step is a set of four categories in which the reviews can be categorised: Information Giving, Information Seeking, Feature Request and Problem Discovery.

Feature Extraction

In this step the authors extracted a set of meaningful features from user reviews data that can be used to train machine-learning techniques and automatically categorise app reviews according to the taxonomy already deduced. They used three techniques: Text Analysis (TA), where they remove the stop words from the reviews, apply stemming to the remaining words and weight words according to their term frequency, Natural Language Processing (NLP) where they identified linguistic patterns and for each one of them they implemented a NLP heuristic to automatically recognise it and Sentiment Analysis (SA), where they assign a quantitative value to a piece of text expressing an affect or mood.

Learning Classifiers

In this step the authors used the TA, NLP and SA techniques to train the machine learning and classify user reviews according to the previously defined taxonomy. They used the Weka tool to automatically classify the reviews and for the machine

learning technique, they used various: The standard probabilistic naive Bayes classifier, Logistic Regression, Support Vector Machines, J48, and the alternating decision tree (ADTree).

Evaluation

In this step the authors evaluated the performance of the machine learning techniques experimented in the previous step. They collected a set of reviews from the apps Angry Birds, Dropbox and Evernote in Apple's App Store, and from TripAdvisor, PicsArt, Pinterest and WhatsApp in Google's Google Play. They manually labelled a total of 1421 sentences of those apps in order to create the training set for the machine learning algorithms. They achieved a precision of 75% of reviews classifications through the J48 algorithm using the NLP, TA and SA techniques.

3.5 The App Sampling Problem for App Store Mining

Authors: William Martin, Mark Harman, Yue Jia, Federica Sarro, and Yuanyuan Zhang

The authors analyse the app sampling problem that happens when researchers use only a subset of apps to study resulting in potential sampling bias. They state that the solution for this problem is very trivial: analyse the complete data set, so you eliminate any possible bias. However, this approach is not always possible because some of the app markets do not provide access to all the historical data (e.g. Google Play, Apple's App Store).

The authors discovered that there is enough evidence that indicates that the partial nature of data available in app stores can pose an important threat to the validity of findings of app store analysis. They say that when the full data set is available, researchers should choose a random subset in order to reduce the bias of the results, but when the full data set is not available, they should argument that the bias does not affect the results (e.g. selecting apps by recency or popularity).

3.6 Identifying Spam in the iOS App Store

Authors: Rishi Chandy, Haijie Gu

The authors aim to relieve the effect that fraudulent reviews (spam reviews) have over mobile apps. For example, those fraudulent could lead users to download potentially harmful apps to their devices or unfairly ignore apps that are being victims of this reviews spam. This review spam on apps can easily be done by users. Some markets have tried to mitigate this problem (e.g. Apple's App Store requires a user to have downloaded the app in order to be able to post a review) but it really did not help at the end, so automatically detecting those spam reviews is a way better to proceed. The goal for the authors is to be able to classify app spam in a supervised setting with limited labelled data, and to cluster those reviews in an unsupervised setting. They also compare the app spam classification using a decision tree model and a novel latent class graphical model. They propose a latent class model with interpretable structure and low complexity, which achieved a significant higher accuracy than using a decision tree model when classifying spam reviews.

4 TOOL

As one of the objectives of the thesis, a web tool was created in order to allow apps' developers to analyse the reviews left by the users of their apps in a friendly way. In particular, the functionalities provided by the tool are:

- Creation of a new user using an e-mail account and a password;
- Creation of new mobile applications for which the user is interested in classifying reviews;
- Importing the user reviews from Google Play Console Center;
- Visualisation, for every application, of the already imported reviews organised by their categorisation and clustering;
- The possibility to vote a single review whether this was correctly categorised or not, and if not, select the right category;
- The option to re-train the machine learner with the reviews that have already been imported and have already been voted by the user.

Figure 1: Tool Main Screen reports a screenshot of the main screen of the tool. There, we can see that the tool has three main parts:

- Top Bar: which contains the logo of the tool on the left, the name of the application being shown on the centre, and the name of the user on the right that, when clicked, will display a menu to the user.
- Left Menu: which contains the list of applications that the user has already created.
- Content Area: Which contains the list of the reviews, previously imported by the user, already categorised and clustered.

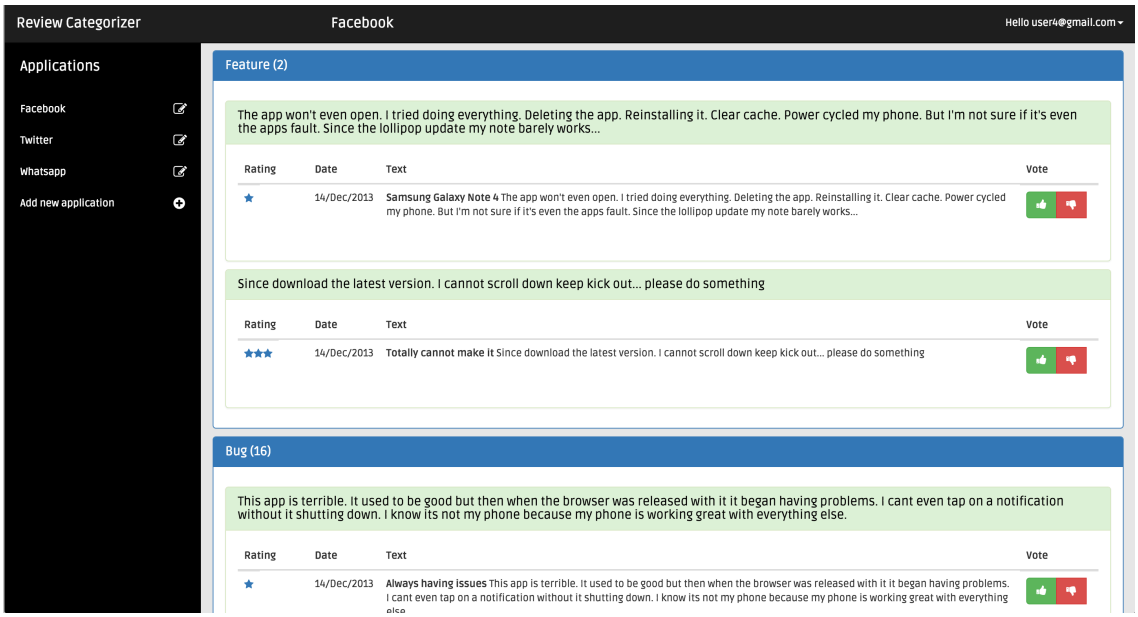

Figure 1: Tool Main Screen

Figure 2 reports an overview of the architecture of the tool. There, we can see that the tool follows a client-server architecture. The Web Client part contains the graphic user interface (GUI) that can be accessed through a web-browser. The server side contains the logic to manage the users and the reviews imported by the users, as well as the logic to (i) re-train the machine learner, (ii) classify reviews into the three categories previously defined and (iii) cluster reviews belonging to the same category. Figure 3 shows the reviews classifier component more in details, which will be described in section 4.1, and Figure 4 shows the cluster reviews component more in details, which will be described in section 4.2.

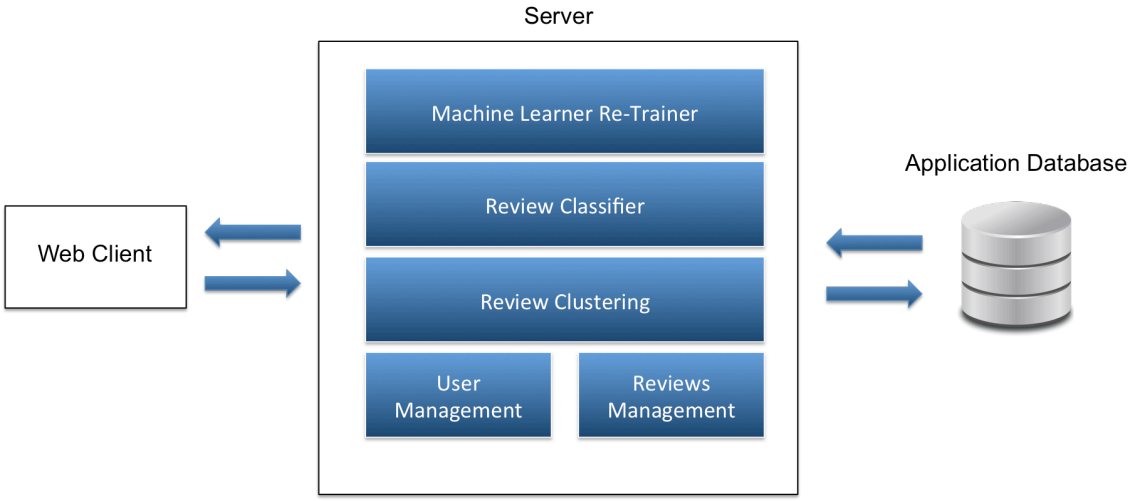

Figure 2: Architecture of the tool

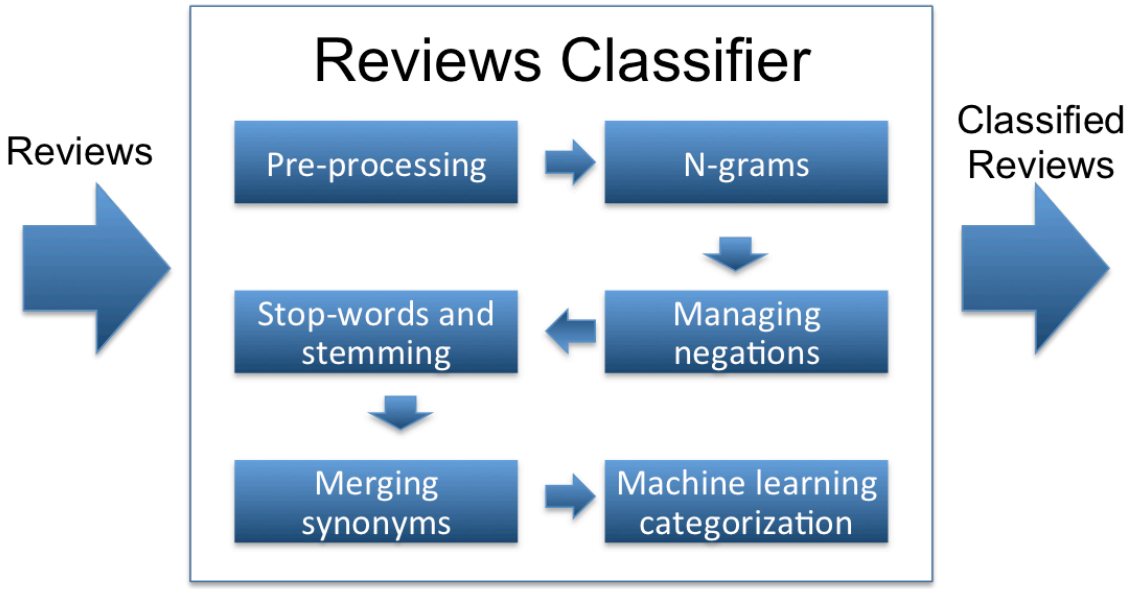

Figure 3: Reviews classifier architecture

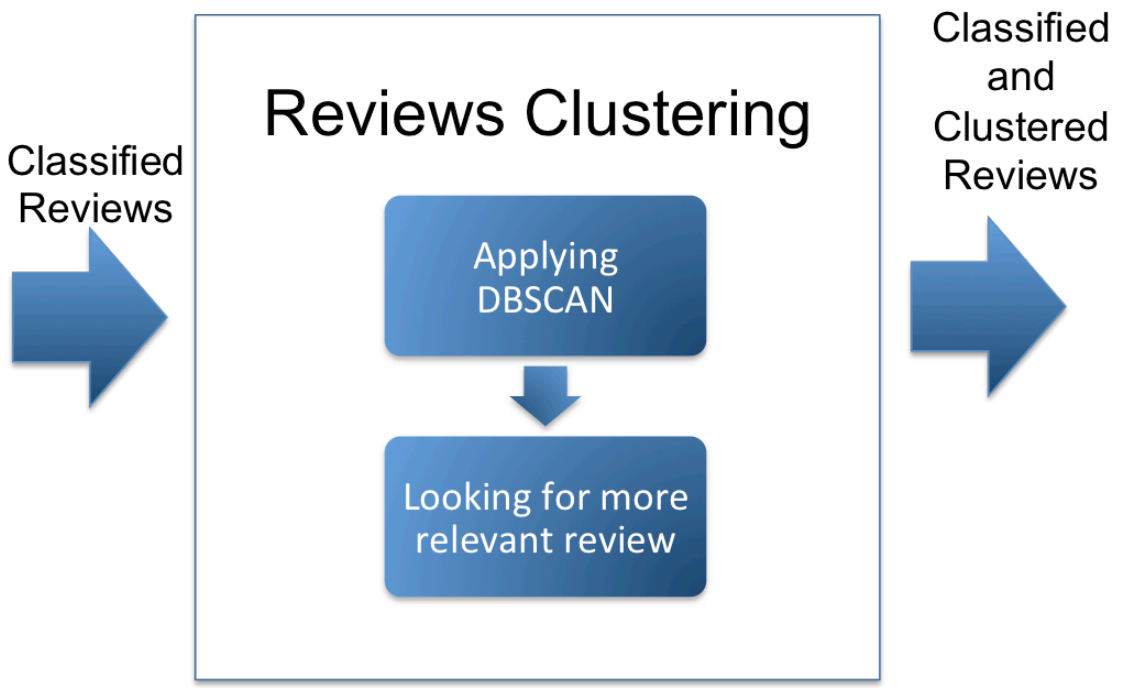

Figure 4: Reviews clustering architecture

4.1 Categorising User Reviews

The goal of this stage is to categorise reviews automatically, using a machine-learning algorithm, into three categories: bugs reporting, suggestions for new features, and other. We are aware that a lot more information can be extracted from the user reviews (e.g. comments about the user experience when using the app) but we chose to focus on bugs reporting and suggestions for new features because this is the most valuable information about how to improve an app from a current version to a next release version. Below we can see a description of the three categories of the reviews:

- **Suggestions for new feature:** A review belongs to this category if in its content it is proposing a new feature
- **Bugs reporting:** A review belongs to this category if in its content it is reporting a bug in the application

- **Other:** A review belongs to this category if its content does not belong to any of the previous categories

In order to be able to determine the categories, we used Weka [Weka, 2014], which is a collection of data mining algorithms for data mining tasks. We exploited decision trees [Breiman et al., 1984] combined with the bagging meta-algorithm [Breiman 1996] in order to classify the reviews into the three categories defined previously. Decision trees are prediction models suitable to solve classification-type problems, where the goal is to predict values of a categorical variable from one or more continuous and/or categorical predictor variables, while the bagging meta-algorithm has been designed to "improve the accuracy of unstable procedures" [Breiman 1996] and consists of splitting the training data into n new training sets, building on each of them a specific classifier (decision tree in our case). When a new instance has to be classified, the n built classifiers vote on the category to which it should be assigned. In this case, the categorical dependent variable is represented by the type of information reported in the review (bugs reporting, suggestions for new features, or others) and we use the rating of the user reviews and the terms/sentences present in them as the predictor variables. While the extraction of the rating assigned in a review is trivial (it is part of the information contained in each user review), we adopt a customised text-normalisation process to characterise each review on the basis of its textual content. This text-normalisation process will be described below.

4.1.1 Pre-processing of reviews

In this step we perform a basic processing of the text. This consists of eliminating all non-alphanumeric character of the review text (such as punctuation signs, etc.) and converting all the words to lowercase. The result of this step is the review text where the words do not contain any non-alphanumeric character, are in lowercase and are separated from each other by a single space.

4.1.2 N-grams extraction

In this step we extract, from each review text pre-processed, the set of n-grams composing it. An n-gram is the composition of n consecutive words in a text. For instance, in the phrase "The app resets itself when opened; Needs to be fixed" an example of a one-gram would be every single word contained in the review, like "The", "app", etc., an example of a two-gram would be every possible composition of two consecutive words, like "app resets", "when opened", etc. We considered the n-grams going from one-gram to four-gram, which, in the previous example phrase, will result in the extraction of n-grams like "resets itself", "Needs to be fixed", etc. The result of this step is the reviews text characterised by the extracted n-grams composing them.

4.1.3 Stop-words and Stemming

In this step we remove the "noise" from the users review text. This noise is composed by the stop-words, which are the most common words in a language ["stop words" Wikipedia, 13], and words being less than three characters long. Also, we apply stemming ["stemming" Wikipedia, 13] to the words in order to reduce all of them to their root. However, noise removing and stemming are only applied to the extracted one-grams of the reviews, which are the words. This is done to avoid the lost of important semantic information embedded in the n-grams. For example, words like "does" and "not" are present in any English stop-words list. By removing them from the three-gram "does not work" we would simply obtain the one-gram "work" which is no longer representing anymore the message of the original three-gram. The result of this step is the reviews text characterised by the extracted n-grams composing them, but removing the noise and applying stemming to the one-grams.

4.1.4 Managing negations

While reviewing the categorisation correctness, we saw that the majority of the user reviews were categorised in a correct way. However, we discovered some situations in which it was categorising some user reviews as bugs when they clearly were not. Some words that we considered as potential indicators as bug reporting (e.g. lag, glitches) when they were negated in a phrase, they were meaning the opposite as a bug request. For example, consider the following user review: "I love it, it runs smooth, no lag or glitches". In this case, it is clear that the presence of the words "lag" and "glitches" do not indicate a bug report in the review. In order to avoid misclassifications with this kind of words, we adopt a set of regular expression to identify these cases and remove from the reviews the matched text. For example, by matching the regular expression no\s+([a-zA-Z]+)\s+or\s+([a-zA-Z]+) we convert "I love it, it runs smooth no lags or glitches" into "I love it, it runs smooth", a set of words better representing the message brought by the review. Note that also this step is only performed on the single words composing the review (e.g. the two-gram "no lags" will still be part of the textual representation of the review).

4.1.5 Merging synonyms

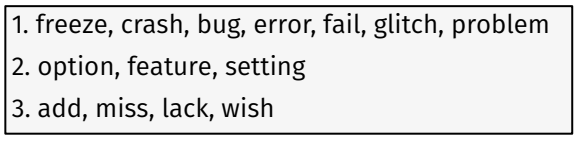

Figure 5: Example of review-related synonyms

Several previous work exploiting textual information for supporting software engineering tasks employ a thesaurus (e.g. WordNet [Miller, 1995]) in order to identify synonyms. For example, in IR-based (Information Retrieval) traceability recovery techniques [Lucia et al., 2007] it is usual to apply synonyms replacement to make sure that textual documents referring to the same concept (e.g. "car") by

using different words (e.g. “car” vs “automobile”) adopt a consistent vocabulary (e.g. the all use “car”), making it simpler for the IR technique to spot related documents. When designing the tool we considered this option. However, reviews for software products, and in particular for mobile apps, contain terms that in standard thesaurus like WordNet are not considered synonyms, while in this specific context indicate the same concept. For this reason, we rely on a customised dictionary of synonyms that we defined manually. Figure 5: Example of review-related synonyms reports an excerpt of the synonyms list we defined [Villarroel et. al, 2015] (words in the same line are considered synonyms). For example, words like freeze, crash, bug, and glitch would not be considered synonyms in a standard thesaurus, while they are very likely to indicate the same concept in mobile apps reviews. Also synonyms’ merging is not applied to n-grams to not break the meaning of sentences (e.g. “the app freezes” should not be converted in “the app bug”).

4.1.6 Creating training set for machine learner

I manually analysed and categorised into one of the previously described categories (e.g. bug reporting, suggestion for new features, and other) 400 reviews. 262 of them were extracted from a backup of Google Play reviews and apps from the 29/04/2012, where 63 were categorised as feature, 109 were categorised as bug and 90 were categorised as other. Also, 138 reviews were extracted from current reviews of open source apps, where 50 were categorised as feature, 38 were categorised as bug and 50 were categorised as other. The overall result was a training set was:

- 113 reviews categorised as feature
- 147 reviews categorised as bug
- 140 reviews categorised as other

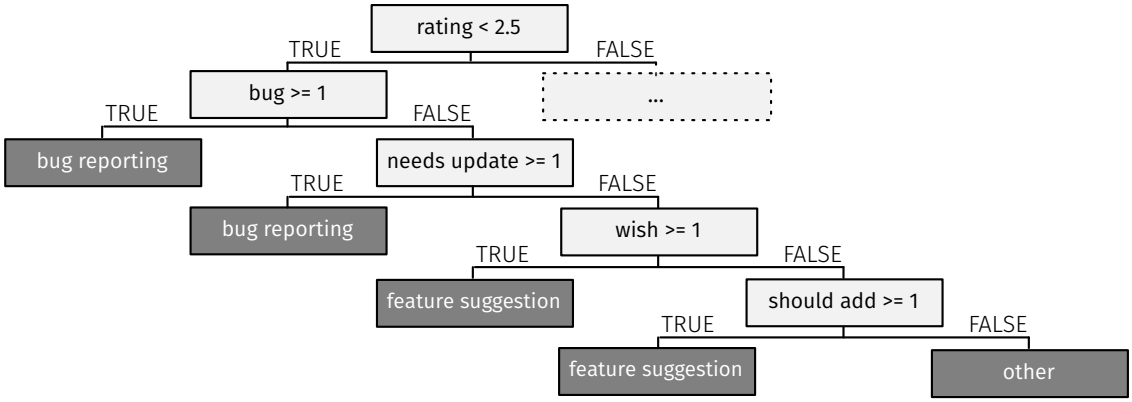

Figure 6: Example of regression tree generated

After the text normalisation process described above each review will be characterised by using as predictor variables: (i) its rating and (ii) the list of n-grams derived from it. Training data, with pre-assigned values for the dependent variables are used to build the decision tree. This set of data is used by the decision tree to automatically select the predictor variables and their interactions that are most important in determining the outcome variable to be explained. The constructed classification tree is represented by a set of yes/no questions that splits the training sample into gradually smaller partitions that group together cohesive sets of data, e.g. those having the same value for the dependent variable. An example of classification tree can be found in Figure 6. Note that we just show one branch of the tree due to lack of space.

As previously mentioned, the tool combines the decision tree method with the bagging meta-algorithm [Breiman, 1996]. It has been designed to "improve the accuracy of unstable procedures" [Breiman, 1996] and consists of splitting the training data into n new training sets, building on each of them a specific classifier (decision tree in our case). When a new instance has to be classified, the n built classifiers vote on the category to which it should be assigned. In our empirical evaluation described in section 5 the use of bagging led to a +3% in classification accuracy. The tool uses the WEKA implementation of decision trees (REPTree class) and bagging (Bagging class).

4.2 Clustering Related Reviews

When the reviews are classified in the three categories described in 4, we cluster reviews inside the same category (e.g. all those in bugs reporting) to identify groups of related reviews (e.g. all those reporting the same bug).

Reviews' clustering is computed by applying DBSCAN [Ester et al., 1996], a density-based clustering algorithm identifying clusters as areas of higher density of elements than the remainder of the data set. This means that it groups together elements that are situated close to each other, assigning the elements in low-density regions to singleton clusters (e.g. clusters only composed by a single element). In the tool, the elements to cluster are the reviews in a specific category and the distance between two reviews r_i and r_j is computed as:

$$dist(r_i,r_j) = 1 - VSM(r_i,r_j)$$

where VSM is the Vector Space Model [Baeza-Yates et Ribeiro-Neto, 1999] cosine similarity between r_i and r_j (e.g. their textual similarity) adopting tf-idf [Baeza-Yates et Ribeiro-Neto, 1999] as terms-weighting schema. Before applying VSM the text in the reviews is normalised by applying the same approach described in section 4.1, with the only exception of the synonyms merging. Indeed, merging synonyms before clustering could be counterproductive since, for example, a review containing "freezes" and a review containing "crash" could indicate two different bugs.

DBSCAN does not require the definition a-priori of the number of clusters to extract, which perfectly matches our case because we cannot know in advance the number of clusters that will be resulted from the reviews. However, it requires the setting of two parameters: (i) minPts, the minimum number of points required to form a dense region, and (ii) epsilon (ε), the maximum distance that can exist between two points (reviews) to consider them as part of the same dense region (cluster). In our approach, we set minPts = 2, since two related reviews are

considered enough to create a cluster in our tool; the value for ε has been empirically defined as detailed in section 5.2.1.

4.3 Running Example

As it was described in section 4, a web tool was created in order to allow users to interact, in a friendly way, with the system of categorisation and clustering of the reviews. One of the typical scenarios, that covers the main functionalities of the tool, is when a user performs the log in, imports a reviews' file that was downloaded from Google Play Developer Console, and then inspects the automatic reviews' classification and clustering presented by the tool. Figure 7: Login shows the login screen of the app. Here, the user must introduce its e-mail and password in order to access the tool. On the other hand, a new user could click on the register button and just create a new user like we can see in Figure 8: Registration.

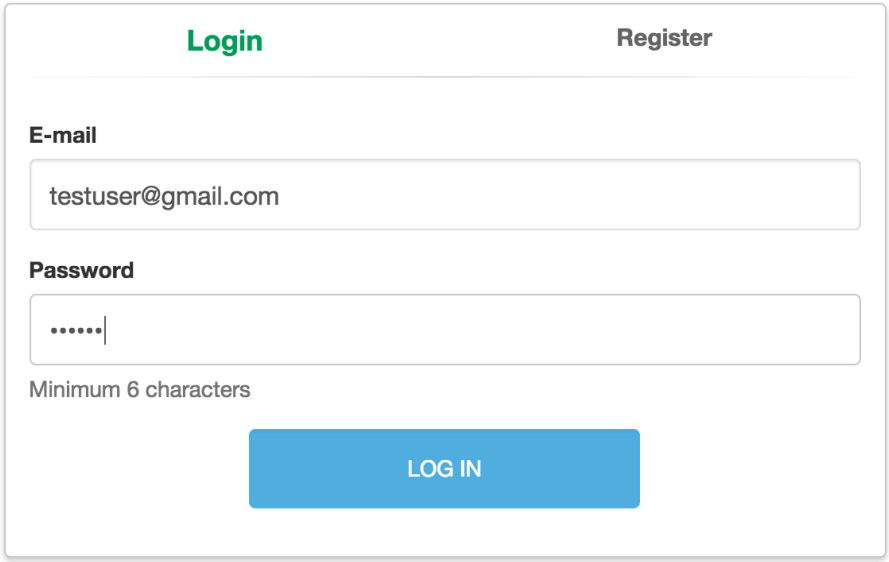

Figure 7: Login

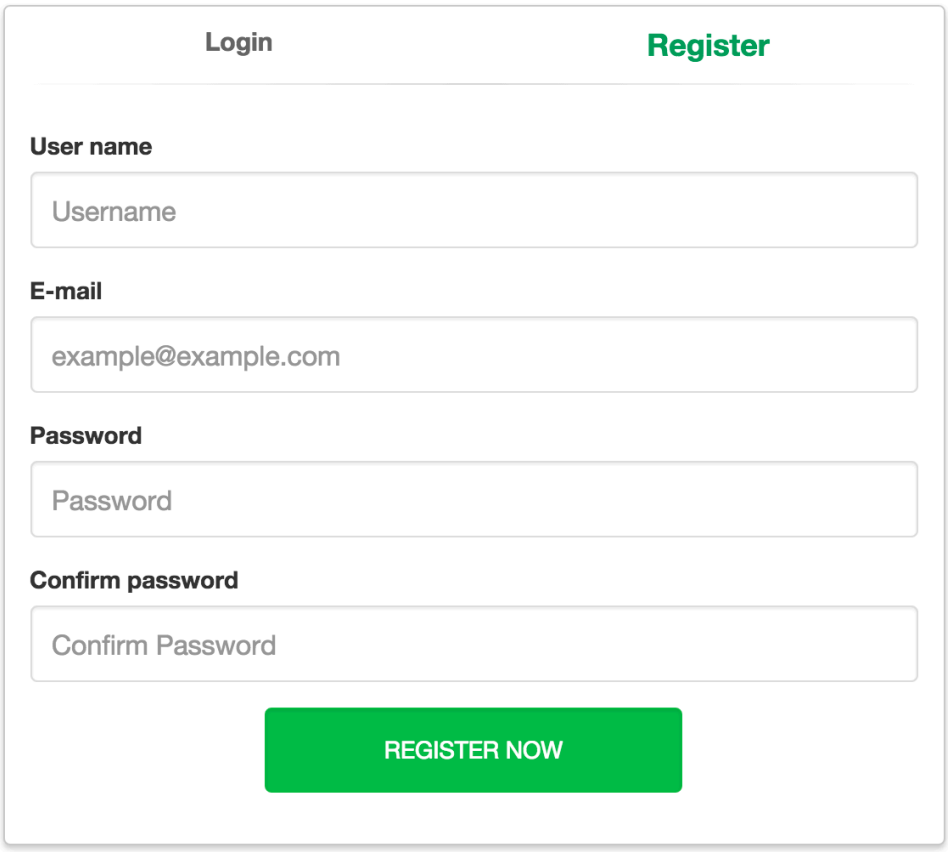

Figure 8: Registration

Once the user is logged in, he will see the main screen of the tool. In Figure 9 we can see the main screen for a fresh user, where it has no app created and no reviews imported yet. From this screen he can select to import a new reviews file or create a new application from the options shown in the content panel depicted Figure 9. He could also select the same options from the dropdown menu that appears when you click the button on the top-right of the tool, like it can be seen in Figure 10.

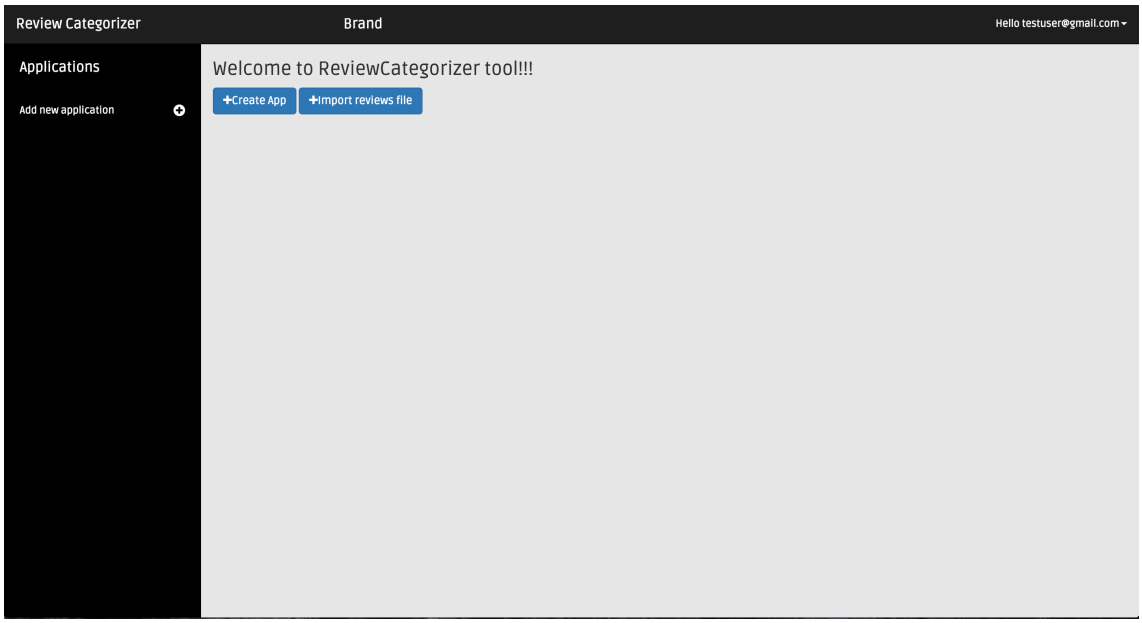

Figure 9: Main screen with fresh user

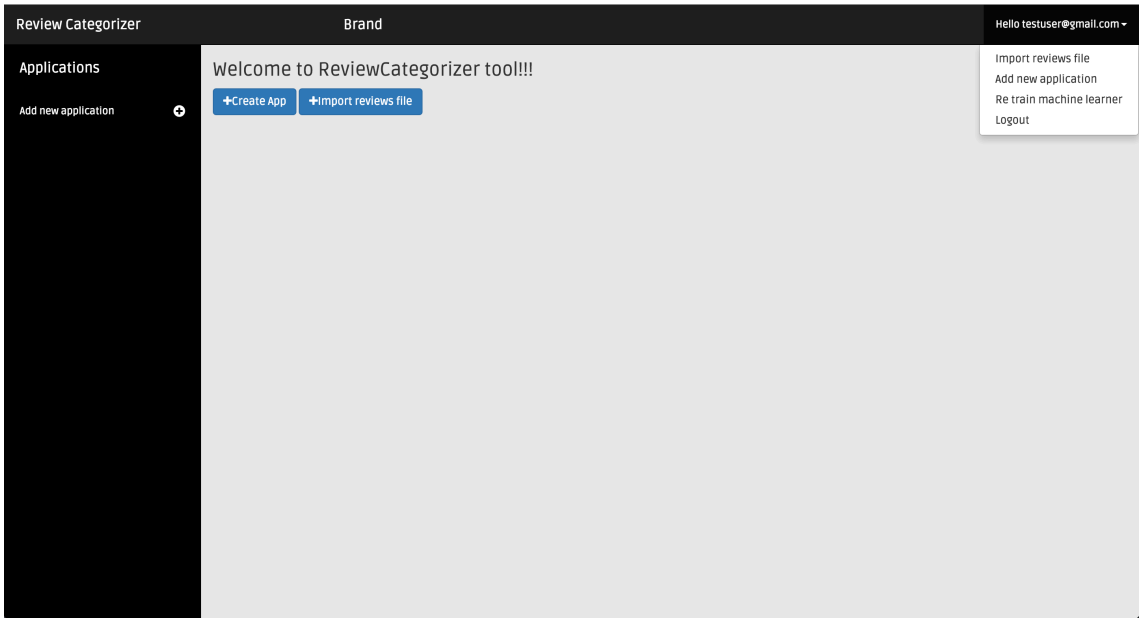

Figure 10: Main screen top right button pressed

When the user presses on “create app” or “add new application” button, then a form is shown with the required fields to create a new application in the tool, as we can see in Figure 11. In addition, there is no need to add a new application before importing a user reviews file. The tool detects automatically to which

application those reviews belong and if application those reviews belong and, if the application does not exist, it is automatically created.

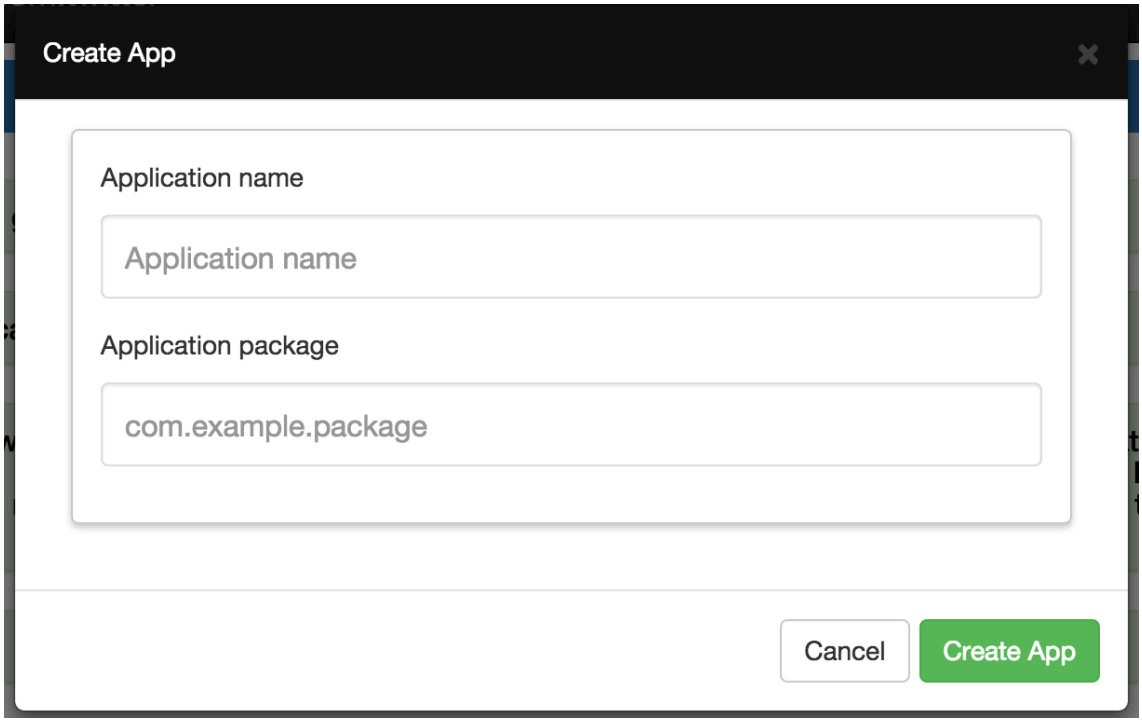

Figure 11: Create application form

When the user clicks the "import reviews file" button, then a file selector form appears where he can choose the csv file that he downloaded previously from Google Play Developer Console. This csv file is a comma-separated file with all the information related to the user reviews of an application in a period of time. In Figure 12 we can see how the tool looks when the user already imported a review file. We can see inside a blue box the name of the categories of the reviews and inside a green box the different clusters or reviews belonging to that category. Furthermore, the user can click any of those clusters in order to expand it and explore which reviews fell inside that cluster, like we can see in Figure 13. Also, each review includes "thumbs up" and "thumbs down" buttons allowing the user to indicate if the review was well categorised or not. If he presses on the thumbs down button, which means that he disagrees with the automatic categorisation of the review, then he will have to tell the tool in which category the review should be

placed instead, as shown in Figure 14. This user feedback on the tools' ability to categorise review aims at improving the performances of the machine learner in charge of categorising the reviews. Indeed, all reviews manually checked (e.g. confirmed or corrected) by the user, will be automatically added to the reviews training set used to categorise new reviews, which should increase in the long term the accuracy of the automatic user reviews categorisation described in section 4.1.

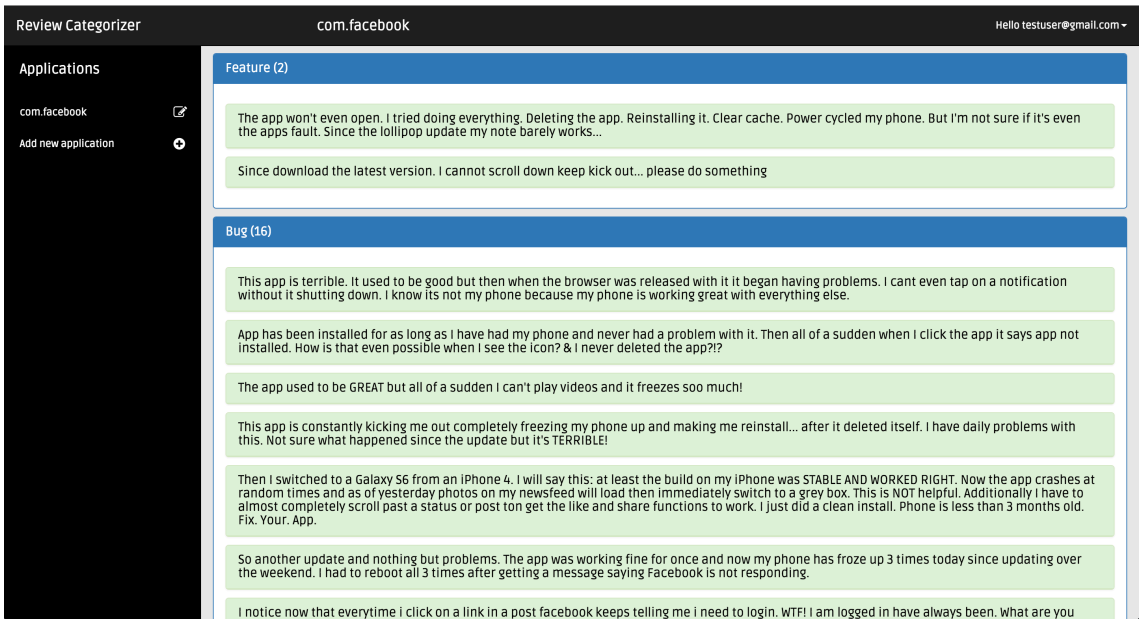

Figure 12: Reviews file imported

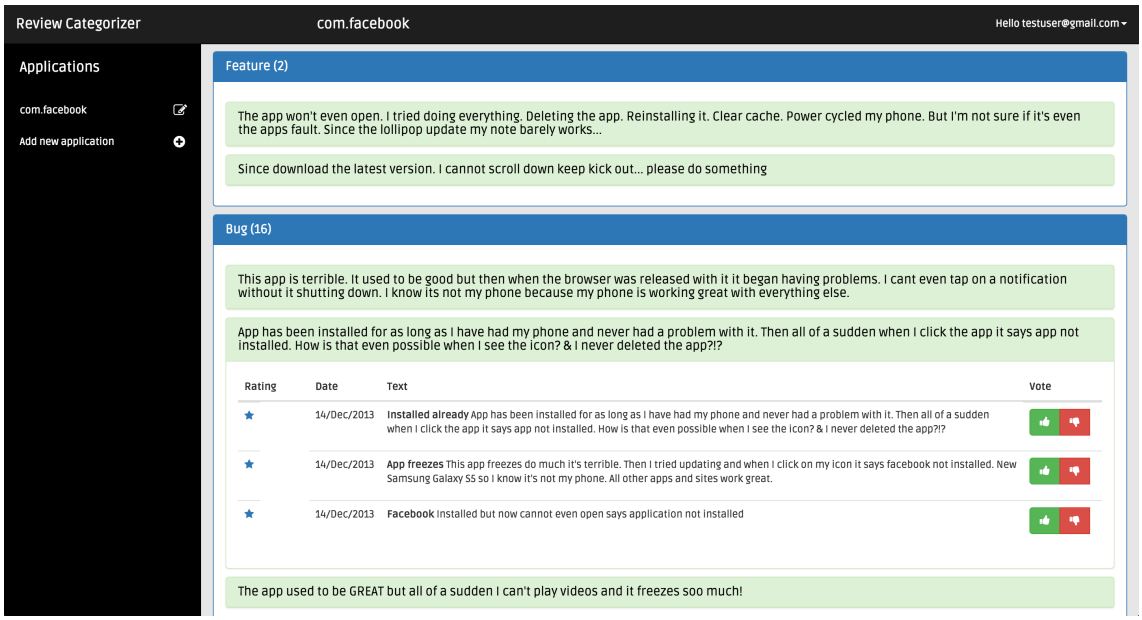

Figure 13: Cluster expanded

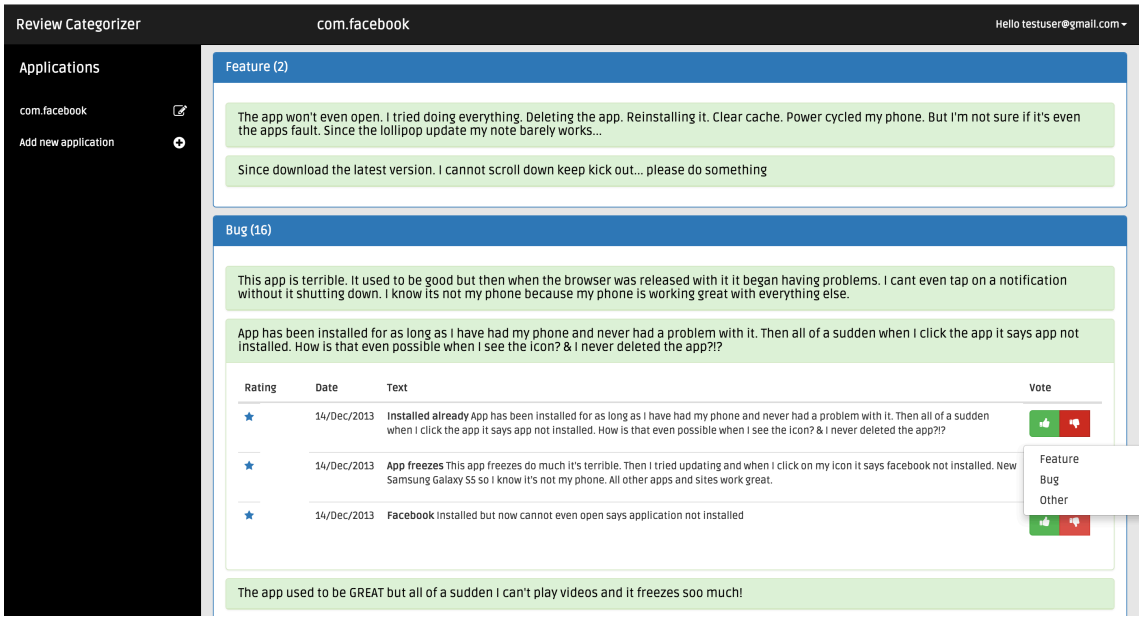

Figure 14: Disagreeing with categorisation

5 EMPIRICAL EVALUATION

In this section we will describe the steps followed to evaluate the tool and its categorisation and clustering processes described in section 4. The process followed to assess the accuracy of the user reviews categorisation will be described in section 5.1 and it will show the performances of our tool in categorising new reviews into the three categories (request for new feature, bug reporting, other) described in section 4.1. The process followed to assess the meaningfulness of the reviews clustering will be described in section 5.2 and it aims at showing how similar the reviews' clusters produced by the clustering process described in section 4.2 are with respect to clusters manually produced by developers. Finally, we started an on-going evaluation in a company developing mobile apps in order to evaluate the tool as a whole (including the categorising and clustering processes and its user interface) in an industrial environment.

5.1 Assessing the Accuracy of the User Reviews Categorisation

In this section we will describe the process followed to evaluate the accuracy of the user reviews categorisation.

5.1.1 Study Design

For this evaluation, we formulated the following research question (RQ):

RQ_1 : *How accurate is the tool in classifying user reviews in the considered categories?*

This RQ aims at evaluating the accuracy of the tool when classifying the reviews into suggestion for new feature, bug reporting and others category, as described in section 4.1.

In order to answer RQ_1 we manually classified a set of 400 reviews randomly extracted from many different Android apps, as described in section 4.1.6. Then,

we used this dataset to perform a 10-fold validation process, consisting of the following five steps:

1. Randomly divide the set of reviews into ten approximately equal subsets;
2. Set aside one review subset as a test set and build the classification model (see section 4.1) with the reviews in the remaining subsets (e.g. the training set);
3. Classify the reviews in the test set using the classification model built on the review training set and store the accuracy of the classification;
4. Repeat this process, setting aside each review subset in turn;
5. Compute the overall average accuracy of the model as the percentage of classified reviews assigned to the correct category. We also report the confusion matrix of the achieved results.

Then, we applied this process 5 times to the set of 400 reviews matching the steps of the text normalisation process described in section 4.1. The first try was with the text of the user reviews with no normalisation applied. Then we included the n-grams extractions to the reviews, and then we followed aggregating the other steps of text normalisation to the reviews (stop words and Stemming, managing negations and merging synonyms). We did that to see whether each individual step had a positive impact on the accuracy of the categorisation of the user reviews or not.

5.1.2 Results Discussion

No text normalisation	N-grams Extraction	Stop words and Stemming	Managing Negations	Merging Synonyms
64%	68%	70%	73%	78%

Table 1: Reviews' Classification Accuracy

	Bug Reporting	Suggestion for New Features	Other
Bug Reporting	119	16	12
Suggestion for New Features	17	89	7
Other	14	24	102

Table 2: Confusion Matrix of Reviews' Classification

As a result of this evaluation, we obtained the accuracy of the categorisation process by applying each step of the reviews' normalisation process as its shown in Table 1. In this table we can see the accuracy achieved by the tool when classifying user reviews in bug reporting, suggestion for new feature and other. In particular, it is shown the accuracy achieved by applying/not applying the different text normalisation that we perform. The first column on the left (No text normalisation) achieved an accuracy of 64% (by providing the machine learner all the words present in the user reviews). By moving toward the right part of Table 1, we can observe the impact on the approach's accuracy when: (i) including the extracted n-grams (+4%=68%), (ii) performing stop words removal and stemming (+2%=70%), (iii) managing the negations via regular expressions (+3%=73%), and (iv) merging synonyms (+5%=78%). Overall, the text normalisation steps adopted in the tool ensure a +14% of accuracy over the baseline of which, the large part, is the result of customised normalisations designed on purpose for our approach.

Table 2 shows the confusion matrix for the categorisation of the reviews with all the normalisation steps applied. Each cell(i,j) represents the number of reviews that belonged to the category represented by the row(i) that were classified with the category represented by the column(j). In this 3x3 matrix, the diagonal represents the number of reviews that were correctly classified according to their pre-assigned category, and the other cells of the table represent the misclassifications of the reviews. As we can see,

310 of the 400 reviews (78%) were correctly classified. It is to mention that classification errors are not concentrated in one of the categories, but the classification accuracy is quite stable across the three categories: 81% for bug reporting, 79% for suggestion for new features and 74% for other.

The most frequent case of misclassification in out tool is the one belonging to other category, for a total of 38 errors that account for the 43% of the overall errors. We looked inside these reviews and in the classification tree' rules generated by our tool to understand the reasons of these misclassifications. In general, we observed that many of those reviews show a low verbosity it making more difficult for the classifier to characterise them from a textual point of view. Furthermore, while reviews reporting bugs and recommending new features are in general characterised by the presence of specific terms (e.g. bug, crash for bug reporting; miss, lack, add for suggestions for new feature) this does not apply for reviews contained in other category.

5.2 Assessing the Meaningfulness of the Reviews Clustering

In this section we will describe the process followed to evaluate the accuracy of the user reviews clustering.

5.2.1 Study Design

For this evaluation, we formulated the following research question (RQ):

RQ_2 : *Are the clusters of reviews generated by the tool meaningful from a developers' point of view?*

This RQ aims on the meaningfulness of reviews clusters extracted by the tool in a specific category of reviews, as described in section 4.2.

In order to answer RQ_2, we asked three industrial developers to manually cluster a set of reviews of three different apps, and then we computed the differences

existing in clusters automatically generated by the tool with the ones manually generated by developers.

We manually collected 160 user reviews among three different Android apps: Facebook, Twitter, Yahoo Mobile Client and Whatsapp. In particular, for each app we collected 20 bug reporting reviews and 20 suggestions for new features reviews, for a total of 40. Then, we asked three industrial developers who work in the same company to manually cluster together the set of reviews belonging to the same category. We explained the developers that the goal was to obtain clusters of reviews referring to the same bug or feature to be implemented. Then, they returned us the manually produced set of clusters (MPSC) that they generated.

Once we received the MPSC, we clustered together the same set of reviews using our tool by the process described in section 4.2. As previously explained, in order to apply the DBSCAN algorithm we needed to tune its ε parameter. We performed such a tuning of the parameter by running the DBSCAN algorithm on the reviews of Yahoo Mobile Client app and varying the ε parameter between 0.1 and 0.9 in intervals of 0.1, for a total of nine configurations (0.0 and 1.0 values were excluded because the output would be a set of singleton clusters and a cluster containing all the reviews, respectively). To define the best configuration among the nine configurations we measured the similarity between the two partitions of reviews (the MPSC and the one obtained running DBSCAN algorithm) by using the MoJo eFfectiveness Measure (MoJoFM) [Wen et Tzerpos, 2004], a normalised variant of the MoJo distance computed as follows:

$$MoJoFM(A,B) = 100 - \left(\frac{mno(A,B)}{max(mno(\forall E_A, B))} \times 100\right)$$

where mno(A, B) is based on the minimum number of Move or Join operations one needs to perform in order to transform a partition A into a partition B, and max(mno(∀ E_A, B) is the maximum possible distance of any partition A from the partition B. Thus, MoJoFM returns 0 if partition A is the farthest partition away

from B; it returns 100 if A is exactly equal to B. The results of this tuning are shown in Figure 15. In general, values between 0.4 and 0.7 allow to achieve very good performances, with its maximum value at 0.6, which is the value that we will use for the tool and for the evaluation.

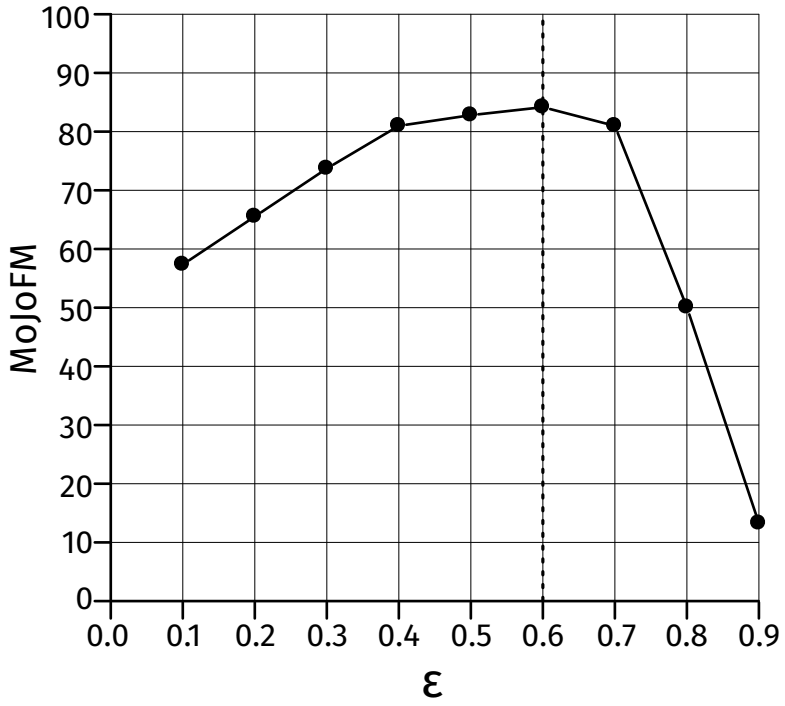

Figure 15: Tuning of the ε DBSCAN parameter

5.2.2 Results Discussion

Table 3 shows the MoJoFM between the clusters obtained from the tool' clustering step (section 4.2) and the clusters that the developers manually reviewed. The MoJoFM achieved is in all cases is higher than 70%, showing a high similarity between the clusters manually and automatically created. In one case, the one concerning clustering suggestion for new features of Whatsapp app, the partitions were exactly the same, indicating the meaningfulness of the clusters resulted by the tool.

	Facebook	Twitter	Whatsapp	Average
Bug Reporting	76%	75%	72%	74%
Suggestion for New Features	73%	83%	100%	85%

Table 3: MoJoFM achieved by the tool

5.3 Evaluating Tool in an Industrial Context

In this section we will describe the process followed to get the opinion of app developers about the tool we built.

We interviewed Giuseppe, a project manager of an Italian software company named Genialapps, and then we lent them the tool in order to try it and transmit us their impressions.

During the interview Giuseppe explained that their most successful app receives hundreds of reviews every week and for a small software company like Genialapps is simply impossible to allocate time to read them all, but they understand that analysing the apps' reviews is "one of the easiest way to increase the rating of your app and, as a consequence, its commercial success". Also, they are aware of the fact that big companies have a team dedicated to extract requirements from user reviews. Our tool can clearly reduce the human resources needed to perform such a manual task.

After having tried the tool, he transmitted us his impressions about the tool. He particularly appreciated the clustering feature, explaining that having all reviews categorised and grouped on the base of their topic saves a lot of time. Also, he recognized the high classification accuracy of the tool when categorizing the app's releases in the three supported categories. Still, not all comments were totally

positive. He gave us very precious feedback about what the industry expects from a tool like the one we created, proposing enhancements to the current interface and new features to implement, which will be taken into account when improving the tool. In the following list we can see what the problems and improvements he suggested are:

- Replace the text that identifies a cluster (currently is the review's text of the most relevant review) with the words overlapped in all the reviews contained in the cluster, which will provide a quicker insight about the content of the cluster.
- There are some problems to understand that the green boxes representing the clusters are actually containing a group of reviews and do not represent a single one.
- Provide a graphical visualization of the "hot trends" in the apps' reviews, in order to see how the topics in the reviews of the apps change over time.
- Implement a prioritization mechanism, highlighting the most important bugs to fix and features to implement.
- Implement an auto update of the reviews, not requiring the manual import from Google Play.
- Implement some filters in the reviews visualization (e.g. all those related to a specific device).
- Keep track of the "implemented" clusters by allowing the developers to mark them in the tool. Additionally, the impact of such implementations can be tracked in terms of user's rating.

Overall, Giuseppe is willing to continue our collaboration and to adopt our tool in his company for a longer case study aimed at fully evaluating it.

5.4 Threats to Validity

Threats to construct validity concern relationships between theory and observation. This threat is generally due to imprecision in the measurements

performed. In our case, this is caused by (i) how the training set for the machine learner was built and (ii) how the manual set of reviews cluster was created. We tried to mitigate this threat by (i) reviewing the manual classification of the reviews of the training set by a second person (in this case, my supervisor) and (ii) by asking three different industrial developers to create the manual set of reviews cluster. Despite this, we cannot exclude the presence of misclassification of the reviews of the training set as well as in the clustering of the user reviews, due to a possible bias of the persons performing those actions.

Threats to internal validity concern factors that could have influenced our results. In our case, this is caused by (i) the use of the MoJoFM as indicator of the similarity between the clusters produced by our tool and those manually created by developers and (ii) the calibration of the ε parameter in section 5.2. The MoJoFM is discussed in [Wen et Tzerpos, 2004] and the calibration of the ε parameter depends directly of the results of the MoJoFM between different partitions of review clusters.

Threats to external validity concern the generalisation of the results. In our case, we always selected reviews from Google Play, but there exist other market of apps (e.g. Apple's App Store, Microsoft's Windows Phone Store, etc.) with may have different type of users, and we have not tested our tool with reviews from those markets so we cannot ensure the accuracy of our classifier with those reviews. In addition, in order to reduce the scope we trained and tested our tool with reviews written in English language. Other languages (e.g. Spanish, Italian, etc.) might contain different keywords and patterns that users use in order to report bugs or suggest new features, so we do not claim that our tool will work well when reviews in different languages than English are being inputted. Furthermore, we arbitrarily selected the different categories (suggestions for new feature, bugs reporting and other) but there might be a wide range of categories that we are not considering, and that it might be interesting for app developers (e.g. positive feedback, reviews

concerning usability problems, etc.) that right now are falling into the *other* category.

6 CONCLUSION AND FUTURE WORK

In this thesis we present a tool that is able to classify user reviews into three different categories (e.g. suggestions for new feature, bugs reporting and other) and, inside the same category (e.g. all those in bugs reporting) identify groups of related reviews (e.g. all those reporting the same bug).

We evaluated the reviews classification process of the tool using a 10-fold validation process over a set of 400 manually classified reviews. Results showed that with the inclusion of all the steps to normalise a review we experienced an increase of the accuracy of the classificator, until reaching the 78% of accuracy, which means that in the majority of the cases the tool correctly classifies the user reviews.

We also evaluated the reviews clustering process of the tool by computing the differences between the clustering of 120 user reviews of three different apps (Facebook, Twitter and Whatsapp) provided by our tool and the ones provided by three industrial developers. Results showed that the clusters created by the tool have, in all cases, a similarity higher than 70% respect to the ones provided by the developers, which shows the meaningfulness of our clustering process.

Finally, we evaluated the tool in an industrial context by interviewing a project manager of a software company and allowing him to try our tool. We discovered that companies of the mobile development sector have a real problem when trying to find useful information in the user reviews because in many cases they cannot afford the work needed to manually process all the user reviews. Our tool is on the right path to relieve companies from this problem, saving them time and increasing the efficiency of reviews processing. Furthermore, it was a very good scenario where to test our tool and its results pointed out the main problems that our tool currently has (i.e. usability problems) and some additional features required by developers in order to process the user reviews more efficiently.

While we tried to ease the work of mobile apps developers when planning the next version of their mobile app, there are still several improvements and new features we need to add to our tool in order to provide a full support. In general, developers want to include the features and bugs that are more important to the users in the new versions of their applications, which means that it would be a good improvement in our tool to rank our classified and clustered reviews and order them by the importance for the users. In addition, the UI of the tool presented in the thesis is a prototype and non-definitive. A usability plan in order to create a user interface that will be more in line to what the developers expect must be developed, which will also include usability testing in order to verify that the new user interface is friendly and it allows developers to use the tool in a more efficient and intuitive way. Additionally, other categories of reviews could be added besides the three already provided by our tool (suggestions for new feature, bugs reporting and other).

7 BIBLIOGRAPHY

["stop words" Wikipedia, 13] Wikipedia: The Free Encyclopedia. Wikimedia Foundation Inc. Updated 22 July 2004, 10:55 UTC. Encyclopedia on-line. https://en.wikipedia.org/wiki/Stop_words

["stemming" Wikipedia, 13] Wikipedia: The Free Encyclopedia. Wikimedia Foundation Inc. Updated 22 July 2004, 10:55 UTC. Encyclopedia on-line. https://en.wikipedia.org/wiki/Stemming

[Ponzanelli et al., 2014] Luca Ponzanelli, Gabriele Bavota, Massimiliano di Penta, Rocco Oliveto, Michele Lanza, Mining StackOverflow to Turn the IDE into a Self-Confident Programming Prompter

[Weka, 2014] http://www.cs.waikato.ac.nz/ml/weka/

[Crider, 2014] Narcissistic Jerks Are Giving Play Store Apps 1-Star Reviews For Higher Visibility http://www.androidpolice.com/2014/01/12/narcissistic-jerks-are-giving-play-store-apps-1-star-reviews-for-higher-visibility/

[Mickel, 2010] The Importance of User Feedback

https://www.pixeltango.com/articles/interaction-design/the-importance-of-user-feedback/

[UserZoom, 2014] 5 Reasons Why User Feedback is Important http://www.userzoom.com/infographic/5-reasons-user-feedback-important/

[JoJo, 2011] How important is feedback? http://ux.stackexchange.com/questions/9472/how-important-is-feedback

[Breiman et al., 1984] L. Breiman, J. Friedman, C. Stone, and R. Olshen. Classification and Regression Trees. Chapman and Hall, 1st edition, 1984.

[Breiman 1996] L. Breiman. Bagging predictors. Machine Learning, 24(2):123–140, 1996.

[Ester et al., 1996] M. Ester, H. peter Kriegel, J. S, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96), pages 226–231, 1996.

[Baeza-Yates et Ribeiro-Neto, 1999] R. Baeza-Yates and B. Ribeiro-Neto. Modern Information Retrieval. Addison-Wesley, 1999.

[Miller, 1995] G. A. Miller. WordNet: A lexical database for English. Commun. ACM, 38(11):39–41, 1995.

[Lucia et al., 2007] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. ACM Transactions on Software Engineering and Methodology, 16(4):13, 2007.

[Villarroel et. al, 2015] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta. Replication package. http://www.inf.unibz.it/~gbavota/reports/app- planning.

[Wen et Tzerpos, 2004] Z. Wen and V. Tzerpos. An effectiveness measure for software clustering algorithms. In Proceedings of the 12th IEEE International Workshop on Program Comprehension, pages 194–203, 2004.